Assignment 3

Due: 04. December 2020, 10:15 CET

General information: This assignment should be completed in groups of **three people**. Submit your assignment via the corresponding LernraumPlus activity. You can only upload one file, therefore make sure to place all your files required to run your solution (.py, .ipynb, .pdf etc) into **one archive** and just upload that archive. For Python assignments you **need to ensure** that your solution can directly be **imported as a Python module** for the automatic tests and at least contains the name of the skeleton file. The easiest way to achieve this, is to just insert your solution into the skeleton .py file directly and submit that.

Python exercise information: Most of the assignments in this course will require you to write Python3 code. Since Python2 is no longer supported, we will only use Python3 in this course.
A general note: Since we want you to understand how the different algorithms work, you are only allowed to use the external libraries that we provide or explicitly mention on the assignment sheet (or that are already within the provided skeleton files).
Exercise information: In this assignment, you will implement functions required for performing independence checks for variables on directed graphs. You will find a reference graph implementation in the *ccbase* package in the *assignment3.zip* alongside the *assignment3.py* skeleton file. The graph implementation can be found in *ccbase/graph.py*, while the used node implementation is located in *ccbase/nodes.py*. The *docs* folder further contains an automatically generated documentation of the provided modules based on the docstrings. The *assignment3.py* file contains the skeletons for the functions that you should implement for this assignment. You are free to use your own graph class from the first assignment instead, however, incorrect behavior of the assignment's tasks due to bugs in your graph class may still lead to point reductions. You also need to make sure to submit all files required for me to run your code!

Exercise 1: (2 Points)

Causal structures like *forks* and *colliders* can be used in a Bayesian network to test for conditional independence given evidence. Implement the following methods:

Task 1: (1 Point) *find_forks(dg)*: Returns all forks of the given directed graph *dg*.

Task 2: (1 Point) *find_colliders(dg)*: Returns all colliders of the given directed graph *dg*.

Exercise 2: (8 Points)

The 4th lecture introduces a first graphical test for independence in Belief Networks called d-separation.

Task 1: (2 Points) Implement the function *get_paths(dg, node_x, node_y)* which computes all *undirected* paths (where each node on the path is only visited once) between node X and node Y in the given directed graph and returns these as a list of lists.

Task 2: (1 Point) Implement *is_collider(dg, node, path)* which returns *True* only if the given node is a collider with respect to the given (undirected) path within the directed graph.

Task 3: (3 Points) Implement the function *is_path_open(dg, path, nodes_z)* that returns *True* only if the given path is *open* (according to d-separation) when conditioned on a (potentially empty) set of variables **Z**.

Task 4: (1 Point) Implement the function *unblocked_path_exists(dg, node_x, node_y, nodes_z)* which checks (and returns *True*) if there exists an unblocked undirected path between two variables X and Y, given a (potentially empty) set of variables **Z**.

Task 5: (1 Point) Implement *check_independence(dg, nodes_x, nodes_y, nodes_z)* which tests if two *sets* of variables **X** and **Y** are conditionally independent given a (potentially empty) variable set **Z**.

You should pay attention to the provided docstrings as they further specify the required behavior of the different functions.

Of course, you are allowed to add any auxiliary functions you find helpful for your implementation.

Hint: You may want to use the functions of earlier tasks when solving later ones. You could hard code solutions for some of these functions for simple examples, if you want to test later ones. Additionally, the automatic tests for each task will also use correct versions of earlier tasks.